Continual Coordination of Spacecraft through Shared Activities

Bradley J. Clement and Anthony C. Barrett
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive, M/S 126-347
Pasadena, CA 91109-8099
{bclement, barrett}@aig.jpl.nasa.gov

Abstract

Interacting agents that interleave planning and execution must reach consensus on their commitments to each other. In domains where agents have varying degrees of interaction and different constraints on communication and computation, agents will require different coordination protocols in order to efficiently reach consensus. ShAC (Shared Activity Coordination) is a framework for designing coordination protocols with an algorithm for continually coordinating agents using these protocols during execution. We show how to construct a wide range of protocols using this framework and describe how ShAC coordinates two rovers and an orbiter in a simulated Mars scenario.

Introduction

When interleaving planning and execution, an agent adjusts its planned activities as it gathers information about the environment and encounters unexpected events, and interacting agents coordinate these adjustments to manage commitments with each other. The work presented here addresses how these agents can interleave coordination with execution. Our ultimate goal is to create interacting agents that autonomously adjust their coordination protocols with respect to unexpected events and changes in communication or computation constraints so that the agents can most efficiently achieve their goals. This paper presents a framework for designing coordination protocols with an algorithm for continually coordinating agents using these protocols during execution.

Our approach, called Shared Activity Coordination (ShAC), provides a general algorithm for interleaving planning and the exchange of plan information based on shared activities. Agents coordinate their plans by establishing consensus on the parameters of shared activities. Figure 1 illustrates this approach where three agents share one activity and two share another. The constraints denote equality requirements between shared activity parameters in different agents. The left vertical box over each planner's schedule represents a *commit window* that moves along with the current time. Activities in this window must be passed on to the execution system, which sends state updates to the planner. Consensus must be established for shared activities before this window to avoid violated commitments between agents. Thus, we introduce the notion of a consensus window (to the right of the commit window) within which consensus must be quickly established before committing. Since consensus is hard to maintain when all agents can modify a shared activity's parameters at the same time, agents must participate in different coordination roles that specify which agent has control of the activity. As shown in the figure, ShAC interacts with the planning and execution by propagating changes to the activities, including their parameters and constraints on the values of those parameters.

ShAC's ability to continually coordinate depends on interleaved planning and execution. As a result, the planner must be able to respond to execution failures and state updates from the execution system. Our implementation interfaces with one such continual planning system, CASPER (Continuous Activity Scheduling Planning Execution and Replanning) (Chien *et al.* 2000). Instead of batch-planning in episodes, CASPER continually adapts near and long-term activities while re-projecting state and resource profiles based on updates from sensors.

First we describe the shared activity model, the ShAC algorithm, and its interface to the planner. Then we specify some generic roles and protocols using the ShAC framework that build on prior coordination mechanisms. Then we describe how our current implementation of ShAC is used to coordinate the communication of two rovers and an orbiter in a simulated Mars scenario. We follow with future research needs revealed in this scenario and comparisons to related work.

ShAC

ShAC is implemented as a module on an agent that commands the agent's planner while communicating with other agents. ShAC keeps track of shared activities and constraints on these activities.

Shared Activities

The model of a shared activity is meant to capture the information that agents must share, including control mechanisms for changing that information. A shared activity is a tuple $\langle$*parameters*, *agent roles*, *protocols*,

The research described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

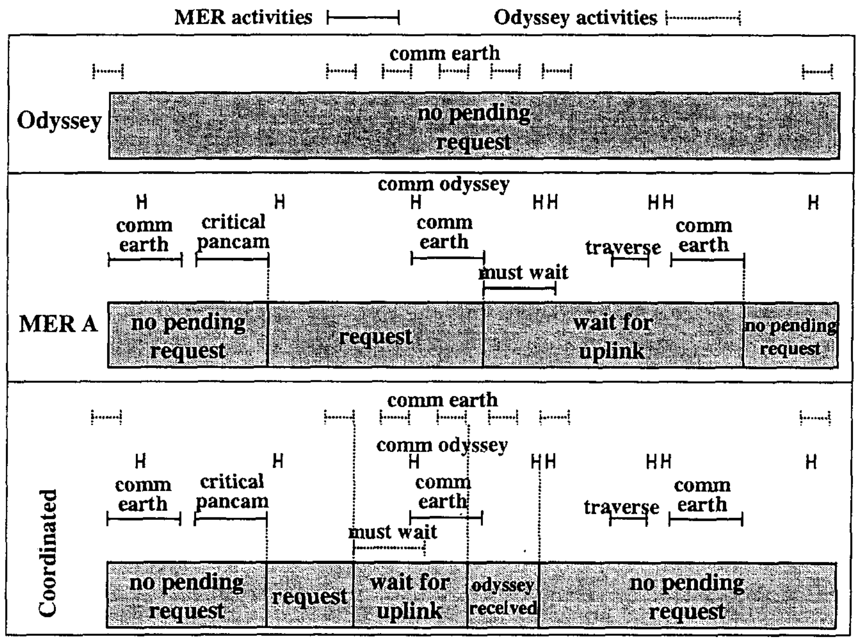

Figure 5: Downlink/uplink shared state for MER A. From top to bottom, Odyssey's initial view, MER A's initial view, and the common view after coordination.

that agents will be able to communicate at all times reliably. In the Mars scenario, the spacecraft communicate with each other in varying time windows and frequencies, and the two MERs can never directly talk to each other. Establishing consensus on beliefs and intentions is impossible without certain communication guarantees (Mullender 1995). Understanding the communication patterns that make consensus possible and the overhead for establishing consensus is critical for multiagent research.

Conclusion

We have introduced shared activity coordination as an approach to designing role-based coordination mechanisms for planning agents. ShAC provides several coordination capabilities upon which we have specified a few higher-level coordination protocols that exercise different aspects of the ShAC model. We have also described an algorithm for continually coordinating planning agents during execution using these protocols. While our future work is aimed at evaluating the benefits of different protocols for different classes of multiagent domains, we validate our approach in coordinating three simulated spacecraft in the presence of an unexpected event.

References

Chien, S.; Knight, R.; Stechert, A.; Sherwood, R.; and Rabideau, G. 2000. Using iterative repair to improve the responsiveness of planning and scheduling. In *Proc. ECP*, 300–307.

Clement, B., and Durfee, E. 2000. Performance of coordinating concurrent hierarchical planning agents using summary information. In *Proc. ATAL*, 213–227.

Corkill, D. 1979. Hierarchical planning in a distributed environment. In *Proc. IJCAI*, 168–175.

Decker, K. 1995. *Environment centered analysis and design of coordination mechanisms.* Ph.D. Dissertation, University of Massachusetts.

desJardins, M., and Wolverton, M. 1999. Coordinating a distributed planning system. *AI Magazine* 20(4):45–53.

Ephrati, E., and Rosenschein, J. 1994. Divide and conquer in multi-agent planning. In *Proc. AAAI*, 375–380.

Georgeff, M. P. 1983. Communication and interaction in multiagent planning. In *Proc. AAAI*, 125–129.

Kraus, S.; Sycara, K.; and Evanchik, A. 1998. Reaching agreements through argumentation: a logical model and implementation. *Artificial Intelligence* 104:1–70.

Lansky, A. 1990. Localized search for controlling automated reasoning. In *Proc. DARPA Workshop on Innov. Approaches to Planning, Scheduling and Control*, 115–125.

Mullender, S. 1995. *Distributed Systems.* Addison-Wesley New York.

Pynadath, D.; Tambe, M.; Cauvat, N.; and Cavedon, L. 1999. Toward team-oriented programming. In *Proc. ATAL*.

Tambe, M., and Jung, H. 1999. The benefits of arguing in a team. *AI Magazine* 20(4).

Tambe, M. 1997. Towards flexible teamwork. *Journal of Artificial Intelligence Research* 7:83–124.

Yokoo, M., and Hirayama, K. 1998. The distributed constraint satisfaction problem: Formalization and algorithms. *IEEE Trans. on KDE* 10(5):673–685.

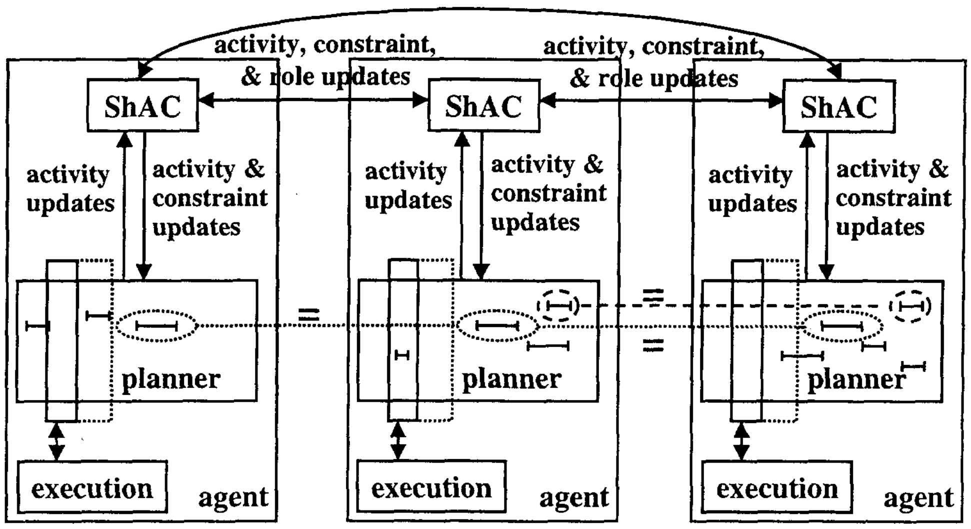
activity, constraint, & role updates
activity, constraint, & role updates
ShAC
ShAC
ShAC
activity updates
activity & constraint updates
activity updates
activity & constraint updates
activity updates
activity & constraint updates
planner
planner
planner
execution
execution
execution
agent
agent
agent

Figure 1: Activities shared among continual planners

decomposition, constraints⟩. The parameters are the shared variables and current values over which agents must reach consensus by the time the activity executes. The agent roles determine the local activity of each agent corresponding to the joint action. To provide flexible coordination relationships, the role activities of the shared activity can have different conditions and effects as specified by the local planning model. The shared parameters map to local parameters in the role activity.

For example, a shared data communication activity can map to a `receive` role activity for one agent and a `send` role activity for another. Shared parameters could specify the start time, duration, transfer rate, and data size of the activity. The data size is depleted from the sender's memory resource but added to the receiver's memory. The agents could have separate power usages for transmitting and receiving. In this case the resources are not shared. Another shared activity could be the use of a common transport resource. Although one agent in an active transit role actually changes position, other agents in passive roles have local activities that only reserve the transport resource. Figure shows an instance of this shared activity where an orbiter receives communication from a rover.

Protocols are the mechanisms assigned to each agent (or role) that allow the agents to change constraints on the shared activity, the set of agents assigned to the activity, and their roles. In Figure , both the orbiter and rover use an argumentation protocol to negotiate the scheduling and attributes of the communication. Constraints will be described in the next section, and a variety of protocols will be defined in the Protocols section.

The shared decomposition enables agents to select different team methods for accomplishing a higher level shared goal. Specifically, the decomposition is a set of shared subactivities. The agents can choose the decomposition from a pre-specified set of subactivity lists. For example, a joint observation among orbiters could decompose into either (`measure`, `process_image`, `downlink`) or (`measure`, `downlink`).

Constraints

Constraints are created by agents' protocols to restrict sets of values for parameters (*parameter constraints*) and permissions for manipulating the parameters, changing constraints on the parameters, and scheduling shared activities (*permission constraints*). These constraints restrict the privileges (or responsibilities) of agents in making coordinated planning decisions. By communicating constraints, protocols can come to agreement on the scheduling of an activity without sharing all details of their local plans.

A parameter constraint is a tuple ⟨*agent*, *parameter*, *value set*⟩. The *agent* denotes who created the constraint. Some protocols differentiate their treatment of constraints based on the agent that created them. For example, the asynchronous weak commitment algorithm prioritizes agents so that lower-priority agents only conform to higher-priority agent constraints (Yokoo & Hirayama 1998). Agents can add to their constraints on a parameter, replace constraints, or cancel them. A string parameter constraint, for example, can restrict a parameter to a specific set of strings. An integer or floating point variable constraint is a set of disjoint ranges of numbers. Scheduling constraints can be represented as constraints on a start time integer parameter. This is shown in Figure where the rover restricts the start time of the communication between two eight minute intervals.

Permission constraints determine how an agent's planner is allowed to manipulate shared activities. The following permissions are currently defined for ShAC:

- parameters - change parameter values
- move - set start time
- duration - change duration of task
- delete - remove from plan

```
shared_activity communicate comm_id_12 {
  time start_time = 2004-302:09:30:00; // date
  int duration =  200;     // seconds
  int data_size = 25600;   // 25.6 Mbits
  real xmit_rate = 128.0;  // 5.0 Kbps
  int priority = 1;        // critical
  roles =
    receive by orbiter,
    send by rover;
  protocols =
    receive argumentation,
    send argumentation;
  permissions =
    receive (move, delete, xmit_rate),
    send (delete, data_size, priority);
  parameter_constraints =
    rover start_time = ([2004-302:09:30:00, 2004-302:09:38:00],
                        [2004-302:18:30:00, 2004-302:18:38:00]);
}
```

Figure 2: An instance of a shared communication activity between a rover and orbiter

- choose decomposition - select shared subactivity of an *or* activity
- add - add to plan[1]
- constrain - send constraints to other agents

In the communication example in Figure , the receiver is allowed to reschedule (move) the activity, delete it, or change the transmission rate. The sender cannot move the activity, but can delete it and change the requested size and priority of the data.

Coordination Algorithm

The purpose of the ShAC algorithm is to negotiate the scheduling and parameters of shared activities until consensus is reached. Figure 3 gives a general specification of the algorithm. ShAC is implemented separate from the planner, so steps 1 through 3 are handled by the planner through an interface to ShAC. Step 4 invokes the protocols that potentially make changes to refocus coordination on resolving shared activity conflicts and improving plan utility. ShAC sends modifications of shared activities and constraints to sharing agents in step 5. In step 6, shared activities and constraints are updated based on changes received from other agents.

Ignoring coordination, a continuous planner must determine when it is appropriate to release activities to the execution system. In some cases, an activity involved in a conflict may either be released (requiring the planner to recover from potential failures) or postponed (to allow the planner to recover before a failure occurs). CASPER keeps a *commit window* (an interval between the current time and some point in the near future) within which activities cannot be modified and passes these activities to the execution system.

[1]This permission applies to a class of shared activities (i.e. an agent may be permitted to instantiate a shared activity of a particular class).

This interaction with the execution system becomes more complicated when agents share tasks. ShAC must make sure that when a shared activity is released, all agents release it while in consensus on the start time and other parameters of the task. Ideally the agents should establish consensus before the commit window. ShAC avoids changes in the commit window by keeping a *consensus window* that extends from the commit window forward by some period specific for the activity. As time moves forward, the windows extend forward. When a shared activity moves into the consensus window, the agents switch to the simple consensus protocol to try and reach consensus before the activity moves into the commit window.

Protocols

In general, protocols determine when to communicate, what to communicate, and how to process received communication. During each iteration of the loop of the coordination algorithm (Figure 3), the protocol determines what to communicate and how to process communication. A protocol is defined by how it implements the following procedures to be called during step 4 of the ShAC coordination algorithm for the shared activity to which it is assigned:

1. modify permissions of the sharing agents
2. modify locally generated parameter constraints
3. add/delete agents sharing the activity
4. change roles of sharing agents

The default protocol, representing a base class from which other protocols inherit, does nothing for these methods. However, even with this passive protocol, the ShAC algorithm still provides several capabilities:

joint intention A shared activity by itself represents a joint intention among the agents that share it.

```
Given: a plan with multiple activities including a set of shared_activities with constraints and a projection of plan into the future.
1. Revise projection using the currently perceived state and any newly added goal activities.
2. Alter plan and projection while honoring constraints.
3. Release relevant near-term activities of plan to the real-time execution system.
4. For each shared activity in shared_activities,
   • if outside consensus window,
     – apply each associated protocol to modify the shared activity;
   • else
     – apply simple consensus protocol.
5. Communicate changes in shared_activities.
6. Update shared_activities based on received communications.
7. Go to 1.
```

Figure 3: Shared activity coordination algorithm

mutual belief Parameters or state assertions of shared activities can be updated by sharing agents to establish consensus over shared information.

resource sharing Sharing agents can have identical constraints on shared states or resources.

active/passive roles Some sharing agents can have active roles with execution primitives while others have passive roles without execution primitives.

master/slave roles A master agent can have permission to schedule/modify an activity that a slave (which has no permissions) must plan around.

The following sections describe some subclasses of this abstract protocol, demonstrating capabilities that each protocol method can provide.

Argumentation

Argumentation is a technique for negotiating joint beliefs or intentions (Kraus, Sycara, & Evanchik 1998). Commonly, one agent makes a proposal to others with justifications. The others evaluate the argument and either accept it or counter-propose with added justifications. This technique has been applied to teamwork negotiation research to form teams, reorganize teams, and resolve conflicts over members' beliefs (Tambe & Jung 1999). It can also be used to establish consensus on shared activities.

A shared activity and associated parameter values are the proposal or counterproposal. Justifications are given as parameter constraints. A proposal is a change to a shared activity that does not violate any parameter constraints. A counterproposal may violate constraints. Protocol method 2 must be implemented to provide the parameter constraint justifications for proposals and counter-proposals. In order to avoid race conditions, protocol method 1 regulates permissions.

Argumentation method 1

```
• if this agent sent the most recent proposal/counterproposal
  – if planner modified shared activity
    * remove self's modification permissions
• else
  – give self modification permissions (e.g. move and delete)
```

Argumentation method 2

```
• if planner modified shared activity
  – generate parameter constraints describing locally consistent values
```

As an example, one agent can propose an activity with a particular start time and add justifications in the form of all intervals within which the shared activity can be locally scheduled. Other agents can replan to accommodate the proposal and counter-propose with their own interval restrictions if replanning cannot accommodate others' constraints. If the agents cannot establish consensus before the consensus window, a higher ranking agent can mandate a time that benefits most of the agents. Of course, there are many variations on this example. Agents may be restricted because they are slaves or do not have constraint permissions to counter-propose.

Delegation

Delegation is a mechanism where an agent in a passive delegator role assigns and reassigns activities to different subsets of agents in active subordinate roles. The delegator and subordinate protocols only need

Delegator method 3

```
• if agent roles empty
  – choose an agent to whom to delegate the activity
  – add (agent, subordinate) to agent roles
```

Subordinate method 3

```
• if cannot resolve conflicts/threats involving activity
  – remove self from agent roles
```

Constraint-Based Conflict Resolution

For this protocol, the agents initially have no permissions to modify a proposed shared activity. They broadcast any parameter constraints to the sharing agents as the planner schedules other local or shared activities around the shared activity while trying to satisfy as many of the others' constraints as possible. After some time period, or once the agents have converged on a set of constraints (not guaranteed), the agents switch to another protocol (e.g. argumentation) potentially reinstating permissions and negotiate final parameter values or delete the activity. The protocol must implement method 2 for generating parameter constraints and method 4 to switch protocols.

Constraint-Based Conflict Resolution method 2

- if cannot resolve conflicts/threats involving shared activity
  - update parameter constraints describing locally consistent values

Constraint-Based Conflict Resolution method 4

- if reached consensus on constraints or $time\_elapsed >$ threshold
  - switch to protocol for resolving conflicts

Centralized Conflict Delegator

Here, a single agent serves in a passive delegator role for a set of shared activities. The delegator models all shared resources and, thus, keeps track of all conflicts for a group of active subordinates. Subordinates do not share activities with each other. The delegator assigns conflicts to different agents by delegating tasks involved in conflicts to different subordinates and also sending the subordinates the corresponding parameter constraints it generates indirectly from the activities it shares with other subordinates. This protocol can subclass from the basic delegation protocol. The difference is in how it chooses the agent to whom to delegate the activity. Below we define this procedure, which is called from Delegator method 3. This function ensures that agents are not modifying the same activities or working on the same conflicts (in order to avoid race conditions).

ChooseSubordinate method

- sort agents in increasing order of times this activity was delegated to them
- for each agent
  - if not delegated any activities involved in conflicts with this one
    - return agent
- return first agent

Application to Mars Scenario

Now we describe how ShAC is applied to a simulated scenario involving two Mars Exploration Rovers (MERs) and a Mars Odyssey orbiter. Different master/slave and active/passive roles are defined using permission constraints for the shared activities to implement a basic protocol for coordinating communication to and from Earth. We will apply some of the previously defined, more sophisticated protocols to this domain in our future work.

The MERs (MER A and MER B) and Odyssey can communicate with Earth directly, but the MERs can optionally route data through Odyssey, which communicates with Earth at a higher bandwidth. The rovers need daily communication with ground operations to receive new goals. The rovers will often fail to traverse to a new target location and cannot proceed until new instructions come from ground operations. In this scenario both MERs must negotiate with Odyssey to determine how to most quickly get a response from ground after sending an image of the surrounding area.

Each MER has a communication state shared with Odyssey that tracks when the image is generated, when it gets to Earth, and when the response from ground operations arrives to the rover. Shared activities for changing the state are shown for different routing options in Figure 4. The rover's activity for generating an image from its panoramic camera changes the state to `request` to communicate its need to downlink and receive an uplink. Activities for sending the image to Earth (either directly or through Odyssey), change the state to a `wait for uplink` state to indicate that the rover will then be waiting for the uplink. Ground operations needs a period of time to generate new commands for the uplink, so if the uplink is received by Odyssey, the state changes to `received` to indicate that now the rover can get the uplink from Odyssey. Once the rover receives the uplink, the state changes back to the normal `no pending request` state. Rover tasks (such as a traverse) need the uplinked data before executing, so it places a local constraint that shared state be `no pending request` during its scheduled interval. There are no shared resources although communication requests from a MER have effects on many local resources of both the MER and Odyssey. All of the shared activities have active master and passive slave roles. MER and Odyssey both take the master role for activities labeled for them in Figure 4.

CASPER planners for each of the MERs and Odyssey first build their three-day plans separately to optimize science data return, resolving any local constraints on memory, power, battery energy, *etc.* The three-day schedules constitute over 600 tasks for each MER and over 1400 for Odyssey with 30 state/resource variables for each MER and 22 for Odyssey.

When coordination begins, the planners send their communication requests to the other planners. Before these updates are received, the initial views of the shared uplink status are shown in Figure 5. The MERs begin with conflicts with their traverse tasks because the uplink has not yet been received from Earth. The coordination algorithm commands the planners to repetitively process shared task updates, replan to resolve conflicts by recomputing the shared state and modifying scientific measurement operations to adjust for the increased power and memory needs, and send task updates. After a minute and a half, MER A, B, and Odyssey agree on routing the downlink and uplink through Odyssey to get the uplinked commands in time for the traversal on

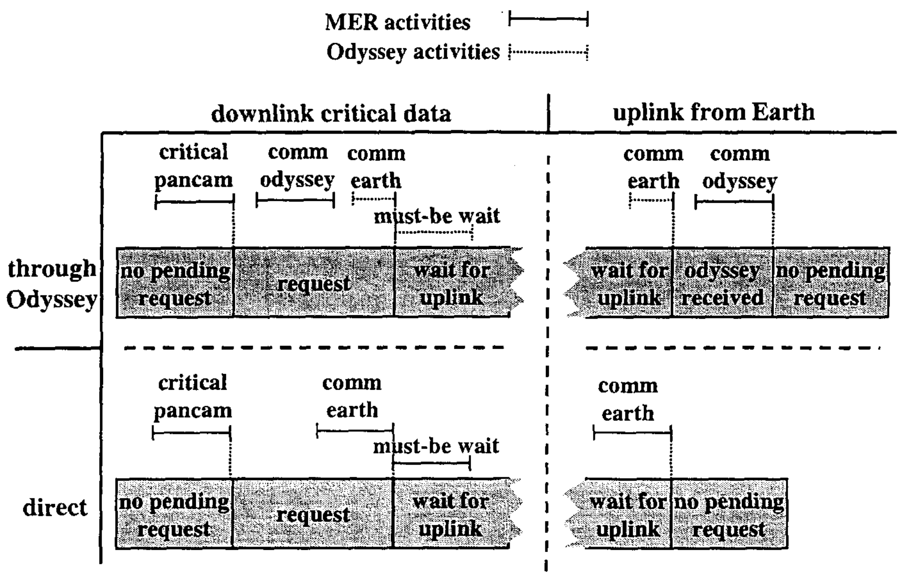

Figure 4: Downlink/uplink states for a rover

different days.[2] The resulting shared state is shown at the bottom of Figure 5. The planners reach consensus that coordination is complete and sleep while waiting for task updates.

Then we triggered an anomaly in MER A's plan causing it to cancel its first day's tasks and shift the entire schedule forward a day. Before sending the updated shared tasks, replanning was issued to resolve local constraints to avoid propagating inconsistent state information to Odyssey. All conflicts were resolved in a few seconds except the traverse conflicts with a `wait` state. Then MER A sends a task update to restart coordination. Coordination completes in less than a minute with data again being routed through Odyssey.

While we have only experimented with simple protocols, this application of ShAC to the Mars scenario shows how planners can coordinate during execution while making minimal concessions to ideal plans and responding to unexpected events. In the next section, we discuss how ShAC builds on related work and discuss new research challenges for decentralized, coordinated planning.

Discussion and Related Work

Conflicts among a group of agents can be avoided by reducing or eliminating interactions by localizing plan effects to particular agents (Lansky 1990), and by merging the individual plans of agents by introducing synchronization actions (Georgeff 1983). In fact, planning and merging can be interleaved (Ephrati & Rosenschein 1994). Earlier work studied interleaved planning and merging and decomposition in a distributed version of the NOAH planner (Corkill 1979) that focused on distributed problem solving. More recent research builds on these techniques by formalizing and reasoning about the plans of multiple agents at multiple levels of abstraction to localize interactions and prune unfruitful spaces during the search for coordinated global plans (Clement & Durfee 2000).

DSIPE (desJardins & Wolverton 1999) employs a centralized plan merging strategy for distributed planners for collaborative problem solving using human decision support. Like our approach, local and global views of planning problem help the planners coordinate the elaboration and repair of their plans. DSIPE provides insight into human involvement in the planning process as well as automatic information filtering for isolating necessary information to share. While our approach relies on the domain modeler to specify up front what information will be shared, ShAC supports a fully decentralized framework and focuses on interleaved coordination and execution.

In many ways this work is following the Generalized Partial Global Planning approach to using a mix of coordination protocols tailored for the domain (Decker 1995). ShAC offers an alternative framework for separating implementation of these mechanisms from the planning algorithms employed by specific agents. Unlike GPGP, ShAC provides a modular framework for combining lower-level mechanisms to create higher-level roles and protocols. Our future work will build on GPGP's evaluations of mechanism variations to better understand how agents should coordinate for domains varying in agent interaction, communication constraints, and computation limitations.

Finally, TEAMCORE provides a robust framework for developing and executing team plans (Tambe 1997; Pynadath *et al.* 1999). This work also offers a decision-theoretic approach to reducing communication within a collaborative framework. Research is needed to investigate the integration of coordinated planning with robust coordinated execution.

An assumption commonly made in multiagent research is

[2]Odyssey's planner ran on a SunBlade 1000, and the MERs ran on a Sparc Ultra 60 and 80.